

Unix : shell scripts

M1101 - Systèmes d'exploitation

Semestre 1, année 2020-2021

2 novembre 2020

Département d'informatique
IUT – Université de Bordeaux

Première partie I

Organisation

- 5 séances de cours intégré
- 3 séances consacrées au projet
- 1 devoir sur table
- 1 projet noté

Deuxième partie II

Rappels, premiers pas, utilisation interactive

Rappels et premiers pas

Quelques commandes

Redirection

Les droits d'accès (bases)

Modification des permissions

Encore des commandes

grep

cut

sort

join

Rappels et premiers pas

Quelques commandes

Vous maîtrisez déjà...

- code, java, javac
- cp, rm, mv, ls
- cat, less, more
- mkdir, rmdir, cd, pwd
- echo, clear
- man
- ...

Redirection

Les commandes produisent du texte sur leur **sortie standard**

Exemples de commandes

```
$ echo "Bonjour"
```

```
$ ls
```

```
$ date "+%H:%M"
```

Ce texte peut être **redirigé** vers un fichier

Exemples : redirections sortie standard

```
$ echo "bonjour" > message.txt
```


```
$ date "+%H:%M" >> message.txt
```

Redirections sortie standard

- Remplacement d'un fichier : 

Exemple

```
$ echo "bonjour" > message.txt
```

- Extension d'un fichier : 

Exemple

```
$ date >> fichier
```

Redirections de la sortie d'erreur

- Certains messages sont produits sur la **sortie d'erreur**

Mise en évidence

```
$ g++ essai.cc > resultat.log  
essai.cc:1: error: ISO C++ forbids declaration of  
'exemple' with no type  
$
```

- Redirection sortie d'erreurs `2 >` `2 >>`

Exemple

```
$ javac MaClasse.java 2> erreurs.txt
```

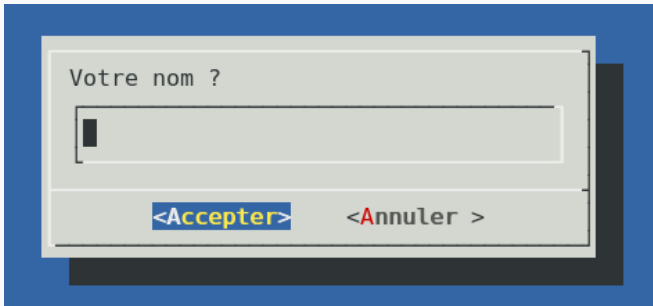
Exemple

```
$ g++ mon-programme.cc 2>&1  
essai.cc:1: error: ISO C++ forbids declaration of  
'exemple' with no type  
$
```


Autres usages de la sortie d'erreur

Exemple

```
$ dialog --inputbox "Votre nom ?" 8 40 2> /tmp/nom
```




Redirection de l'entrée standard

- Depuis un fichier : 

Exemple

```
$ tr '[a-z]' '[A-Z]' < texte.txt
```

- “here document” : 

Exemple

```
$ tr '[a-z]' '[A-Z]' <<XXX
```

```
ceci Est un
```

```
exemple
```

```
XXX
```


Redirection entre deux commandes

L'opérateur `|` (*pipe*) redirige la **sortie standard** d'une commande vers **l'entrée standard** d'une autre commande

Exemple

```
$ w | cat -n
```

On peut constituer un **pipeline** de plusieurs commandes

Exemple : redimensionner une image

```
anytopnm dscn3214.jpg |  
    pnmscale -width 100 |  
        pnmtopng > statue-hcm-100px.png
```

Les droits d'accès (bases)

La commande `ls -l` montre les **droits d'accès**

Exemple

```
billaud@feathers:~/Essais/C++$ ls -l
total 64
drwxr-xr-x 2 billaud profs 4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs 6495 nov 2 21:19 initTab
-rw-r--r-- 1 billaud profs 236 nov 2 21:19 initTab.cc
...
```

Premier caractère

- `d` pour les répertoires (*directory*)
- `-` pour les fichiers

Exemple

```
drwxr-xr-x 2 billaud profs 4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs 6495 nov 2 21:19 initTab
-rw-r--r-- 1 billaud profs 236 nov 2 21:19 initTab.cc
```

Lettres suivantes : indiquent les **droits d'accès** Présentation par groupe de trois :

-

rwX	r-x	r-x
-----	-----	-----

- | |
|-----|
| rwX |
|-----|

 pour le **propriétaire** du fichier (billaud)
- | |
|-----|
| r-x |
|-----|

 les utilisateurs du **groupe profs**
- | |
|-----|
| r-x |
|-----|

 pour les autres

Les lettres indiquent les droits d'accès (ou **mode**, ou **permissions**)

- **r** pour **read** (droit de lecture)
- **w** pour **write** (droit d'écriture, modification)
- **x** pour
 - **execute** (droit d'exécution) pour les fichiers,
 - **x=cross** (droit de traverser) pour les répertoires.

Exemples :

Exemple

```
drwxr-xr-x 2 billaud profs 4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs 6495 nov 2 21:19 initTab
-rw-r--r-- 1 billaud profs 236 nov 2 21:19 initTab.cc
```

- `iniTab.cc` peut être lu et modifié par son propriétaire (`rw`), lu par les membres du groupe (`r`) et par les autres (`r`).
- `a.out` peut être lu, modifié et exécuté par son propriétaire (`rw`), lu et exécuté par les utilisateurs du groupe (`rx`) ainsi que par les autres (`rx`).
- le répertoire `Heap` peut être lu, modifié et traversé par le propriétaire (`rw`), lu et traversé par les membres du groupe (`rx`) et les autres.

Modification des permissions

chmod : changer les droits d'accès

La commande «`chmod`» change les droits d'accès (**CH**ange **MO**De)

Notation octale

Exemple : `chmod 750 mon-fichier`

- chaque groupe de trois bits est codé en octal, avec $r=4$, $w=2$, $x=1$.
- Donc `chmod 750 ...` donne les droits

<code>rw</code>	<code>x</code>	<code>r</code>	<code>x</code>	<code>--</code>
-----------------	----------------	----------------	----------------	-----------------

 au fichier

Exercice : notation octale

Complétez la table d'équivalence

octal	droits	octal	droits
0	- - -	4	
1		5	r - x
2		6	
3		7	r w x

Exercice : droits sur les fichiers

1. Tapez

```
$ ls -l > mon-fichier
```

```
$ chmod 777 mon-fichier
```

- pouvez-vous lire le fichier
(cat mon-fichier)?
- le modifier
(echo >> mon-fichier)?

2. même question après

```
chmod 666 mon-fichier
```

3. ...

droits	lire ?	modifier ?
000	non	non
111		
222		
333		
444		
555	oui	oui
666		
777		

Exercice : droits

1. Tapez

```
$ ls -l > mon-fichier
```

```
$ chmod 777 mon-fichier
```

- pouvez-vous lire le fichier ?
- le modifier ?

2. Changez les droits

```
$ chmod 077 mon-fichier
```

- pouvez-vous le lire ?
- le modifier ?

3. Conclusions ?

Exemple

`chmod u=rwx,g=rx,o= mon-fichier`

- `u` user (propriétaire)
- `g` groupe
- `o` others (autres)

chmod : modification des droits

Sous forme symbolique, permet de **modifier** *certain*s droits.

- + ajouter des droits
- - enlever

Exemple

```
chmod go-w f
```

enlève le droit w au groupe (g) et aux autres (o), sans changer les autres permissions.

Exemple

```
chmod +x f
```

ajoute les droits "x"

Chaque fichier appartient à un utilisateur et à un groupe. Il est possible de les changer avec **chown** et **chgrp**. Des droits d'administration peuvent être nécessaires.

```
rgiot@info-bombarde /tmp $ ls -l test.txt
-rw-r--r-- 1 rgiot utilisateurs du domaine 6 nov.   6 13:34 test
rgiot@info-bombarde /tmp $ chgrp info_personnels test.txt
rgiot@info-bombarde /tmp $ ls -l test.txt
-rw-r--r-- 1 rgiot info_personnels 6 nov.   6 13:34 test.txt
```

Encore des commandes

Quelques commandes

- `grep` : sélection de lignes
- `cut` : sélection de colonnes
- `sort` : tri
- `join` : jointure

la commande grep

Sélectionne des lignes d'un texte, qui contiennent un certain motif (expression régulière)

exemple

- Soit le fichier `villes.txt` accessible sur moodle

```
france:paris
```

```
vietnam:ho chi minh
```

```
italie:roma
```

```
france:bordeaux
```

```
vietnam:hanoi
```

```
inde:delhi
```

- la commande

```
$ grep italie villes.txt
```

affiche les lignes du fichier qui contiennent `italie`

villes.txt

france:paris

vietnam:ho chi minh

italie:roma

france:bordeaux

vietnam:hanoi

inde:delhi

Essayer

```
$ grep 'i' villes.txt
```

```
$ grep ':h' villes.txt
```

grep : expression régulière / ancrage

Les caractères ^ et \$ servent à “**ancrer**” le motif de recherche

- au début d'une ligne
- ou à la fin de la ligne

Essayer

```
$ grep '^i' villes.txt
```

```
$ grep 'i$' villes.txt
```

Les expressions régulières de grep utilisent des méta-caractères qui ont un sens différent de ceux du shell. Voici les essentiels :

- `.` : n'importe quel caractère
- `*` : 0 ou plusieurs fois le motif précédent
- `+` : 1 ou plusieurs fois le motif précédent

La commande cut sélectionne une *colonne* de données.

Essayer

```
$ cut -c 1-3 villes.txt
```

```
$ cut -d: -f1 villes.txt
```

```
$ grep vietnam villes.txt | cut -d: -f2
```

la commande sort

Ordonne les lignes selon un critère

```
$ sort villes.txt
```

```
$ sort -t: -k2 villes.txt
```

Exercice

- Créer un fichier `villes-pays.txt` (villes ordonnées par pays)
- Créer un fichier `continents-pays.txt` (continents ordonnées par pays) à partir de `continents.txt` (également disponible sur moodle)

`europe:france`

`europe:italie`

`asie:vietnam`

`asie:chine`

Commande “join”

Rapproche deux fichiers sur une **clé commune**.

Les fichiers doivent être triés

Exemple

```
$ join -t: -1 2 -2 1 continents-pays.txt villes-pays.txt
```

- `-t:` : délimiteur de champs
- `-1 2` : clé du premier fichier = second champ
- `-2 1` : clé du second fichier = premier champ

Troisième partie III

Shell-scripts : introduction

Qu'est-ce qu'un shell ?

Utilisation interactive / scripts

Pourquoi écrire des scripts ?

Qu'est-ce qu'un shell ?

Qu'est ce qu'un *shell*

«shell» : programme qui

- lit des lignes de commandes
 - tapées par l'utilisateur
 - ou lues depuis un fichier
- les fait exécuter

Autre nom : interprète de commandes

De nombreux shells sont disponibles sous Unix/Linux,

- sh
- **bash** (Bourne again shell),
- CSH (C Shell),
- KSH (KORN Shell),
- TCSH
- ...

Ils

- jouent le même rôle,
- ont des **syntaxes** différentes,
- fournissent des **fonctions prédéfinies** différentes.

- Le fichier `/etc/shells` contient la liste des shells disponibles

```
$ cat /etc/shells
```

- pour savoir quel shell vous utilisez, tapez

```
$ echo $SHELL
```

- Pour connaître votre **shell par défaut**

```
$ finger -l
```

Utilisation interactive / scripts

Usage interactif

1. vous tapez une commande
2. le *shell* l'interprète
3. vous retournez à l'étape 1

Scripts

1. vous tapez des commandes dans un fichier texte (**script**)
2. vous demandez l'exécution de ce script

Shell script

Définition : Shell-script = fichier texte qui contient une suite de commandes.

Exemple : fichier “premier.sh”

```
1  #!/bin/bash
2  #
3  # Mon premier essai
4  clear
5  echo -n "Nous sommes le "
6  date
7  echo "et c'est mon premier script"
```

Note : le caractère # indique un commentaire

Shell script

Exemple : fichier “premier.sh”

```
1  #!/bin/bash
2  #
3  # Mon premier essai
4  clear
5  echo -n "Nous sommes le "
6  date
7  echo "et c'est mon premier script"
```

Exécution par

```
$ bash premier.sh
```

ou

```
$ chmod +x premier.sh
```

```
$ ./premier.sh
```

Pourquoi écrire des scripts ?

Pourquoi écrire des scripts ?

Intérêt

- Permettent de réutiliser des suites de commandes sans risque d'erreur
- Gagner du temps
- Automatiser les tâches fréquentes
- ...

Applications des scripts

- Créer ses propres commandes à partir de commandes existantes
- Scripts d'installation
- Fonctionnement du système d'exploitation (ex : lancement de script au démarrage)
- Aide à l'administration du système (tâche répétitives)
exemple :surveillance des quotas
- ...

Quatrième partie IV

**Variables, paramètres,
expressions...**

Variables

Environnement, export

Tableaux

Paramètres positionnels

Plus d'informations sur l'affectation

Lecture de variables

Essais

read, exercice

read, IFS

Expansions

Exercices

Une petite application

Affectation du résultat d'une commande

Chaînes et expansion

Variables

Les variables du *shell* mémorisent des chaînes de caractères.

- la liste des variables est affichée par **set**
- Variables système définies automatiquement (**et à connaître IMPÉRATIVEMENT**) :
HOME PWD SHELL USERNAME HOSTNAME PATH LANG
etc.

- affectation de variable : `NOM=CHAINE`
 - Ne surtout pas mettre d'espace avant et après =
- Accès au contenu par

`$NOM` # *peut être ambiguë*

`${NOM}` # *moins ambiguë*

Exemple

```
message="Bienvenue parmi nous"
```

```
echo $message
```

Variables : exemple avec affectation temporaire

```
1  #!/bin/bash
2
3  france="Europe/Paris"
4  vietnam="Asia/Ho_Chi_Minh"
5
6  echo "Bonjour  $USER"
7
8  echo -n "heure France = "
9  TZ=$france date
10
11 echo -n "heure Vietnam = "
12 TZ=$vietnam date
13
14 echo -n "aujourd'hui "
15 date
```

Essayez

Avec la commande `set` et `grep`, trouvez les variables qui indiquent

- le nom de votre poste de travail,
- son type,
- la version du système d'exploitation ?

Environnement, export

- un processus s'exécute dans un environnement (valeurs de variables). La commande **printenv** affiche les variables qui sont transmises aux processus.
- les variables système sont transmises (**exportées**) automatiquement lors d'un appel de script
- les autres doivent être exportées explicitement (= ajoutées à l'environnement) :

```
export NOM[=VALEUR]
```

Quelques variables utiles

- EDITOR : éditeur de texte par défaut
- PAGER : pageur par défaut
- PATH : liste des chemins de recherche des exécutable
- PS1 : définition du prompt
- VISUAL : idem que EDITOR / avait un sens il y a 30 ans

Tableaux

L'interprète `bash` possède des `tableaux`

Exemple

```
declare -a pays
```

```
pays[0]=Australie
```

```
pays[1]=Vietnam
```

```
pays[2]=France
```

```
pays[3]=Allemagne
```

```
i=3
```

```
j=1
```

```
echo football :  ${pays[$i]} contre ${pays[$j]}
```

Paramètres positionnels

Paramètres positionnels

Invocation d'un script

Un script peut être **invoqué avec des paramètres**, exemple :

```
./mon-script Hanoi Paris Bordeaux "Ho Chi Minh City"
```

Pendant l'exécution

- \$1="Hanoi",
- \$2="Paris",
- \$3="Bordeaux",
- \$4="Ho Chi Minh City"

Autres variables utiles

- `$#` = 4 – le nombre de paramètres
- `$*` = Hanoi Paris Bordeaux Ho Chi Minh City
- `$@` = Hanoi Paris Bordeaux "Ho Chi Minh City"
- `$0` = `"/mon-script"` – le nom du script

Gestion des arguments supplémentaires

```
$ help shift
```

```
shift: shift [n]
```

Décale des paramètres de position.

Renomme les paramètres de position $\$N+1, \$N+2 \dots$ à $\$1, \$2 \dots$ donné, il est supposé égal à 1.

Code de retour :

Renvoie le code de succès à moins que N soit négatif ou supé

Exercice

Ecrire un script à un paramètre qui indique les villes d'un pays.

```
$ villes.sh france  
paris  
bordeaux  
...  
$
```

Utiliser le fichier de données précédent.

Solutions à discuter

```
#!/bin/bash
```

```
grep "^$1:" villes.txt | cut -d: -f2
```

```
#!/bin/bash
```

```
FICHIER="villes.txt"
```

```
PAYS="$1"
```

```
grep "^${PAYS}:" "${FICHIER}" | \  
cut -d: -f2
```

**Plus d'informations sur
l'affectation**

Met une valeur dans une variable.

Noms de variable

- lettres, des chiffres, blancs soulignés
- ne commence pas par un chiffre
- MAJUSCULES/minuscules différenciées

Affectation (suite)

Deux formes

- `NOM=CHaine`
affectation simple
- `let NOM=EXPRESSION`
affectation du résultat d'un calcul arithmétique

`let` est une commande interne du *bash* (`help let`)

Essayez

```
a=12
```

```
b=42
```

```
c=a+b
```

```
echo $c
```

```
d=$a+$b
```

```
echo $d
```

```
let e=a+b
```

```
echo $e
```

Lecture de variables

La commande

```
read v1 v2 ...
```

lit une ligne au terminal, et affecte les mots dans les variables citées.

Exemple

```
read nom prenom
```

Essayez

- `read nom prenom`
- que se passe-t-il si on tape plus de mots qu'il n'y a de variables ?

Exercice

Écrire un script qui

- demande l'année de naissance,
- affiche l'âge (l'année courante est inscrite en dur dans le script).

Exemple d'exécution

```
$ ./quel_age
```

```
Votre année de naissance ?
```

```
1984
```

```
Vous êtes né en 1984, vous avez donc 33 ans.
```

La variable IFS

indique le séparateur reconnu par read (**input field separator**)

```
$ export IFS=,  
$ read NOM PRENOM  
einstein,albert  
$ echo $PRENOM  
albert
```

Alternative à “export”

Affectation temporaire :

```
IFS=, read NOM PRENOM
```

valide pendant la durée d'exécution du read

Expansions

Définition

Expansion : remplacement d'une expression par sa **valeur**

Exemples

- expansion de **meta-caractères** :
`ls *.tex`
- expansion de **variables** :
`echo bonjour $USER`
- expansion du **résultat d'une commande** :
`echo il y a $(who | wc -l) connexions`
- expansion **numérique** :
`echo périmetre = $((2*(longueur+hauteur)))`

Autre notation

Historiquement, **sh** utilisait des “anti-quotes” pour `$(...)` :

```
echo il y a `who | wc -l` connexions
```

Moins lisible, risque de confusion avec les apostrophes

```
echo il y a 'who | wc -l' connexions
```

1. Exercice simple

Dans le script qui calcule l'âge, remplacez la **constante de l'année courante** par un appel à date `+%Y`

2. Mieux

Écrire un script qui affiche le nombre de processus qui vous appartiennent.

Exécution :

Sur tuba, adupont a 45 processus

Indication : comptez les lignes qui commencent par votre nom dans le résultat de “ps axu”.

Les étudiants ayant un login trop long pour ps (affiché partiellement avec +) doivent utiliser la commande suivante à la place :

```
ps axo user:20,pid,%cpu,%mem,vsz,rss,tty,stat,start,time,comm
```

3. Encore plus fort

Script qui affiche le nom en clair d'une personne (dont on connaît l'identifiant). Utiliser la commande

```
ldapsearch -x cn=adupont displayname
```

Exercices (suite)

Écrire un script qui indique les 5 plus gros sous-répertoires d'un répertoire donné.

Exécution

```
$ plus-gros.sh ~/Essais
196      /home/billaud/Essais/LATEX
152      /home/billaud/Essais/C++
96       /home/billaud/Essais/Python
36       /home/billaud/Essais/PHP
```

Indications

- script à 1 paramètre
- du `-s repertoire/*`
- tri *numérique*
- commande `tail -n nombre`

Une petite application

Application : carnet de téléphone

Sous forme de trois commandes

- `$ tel-ajouter numero nom`
- `$ tel-chercher nom`
- `$ tel-afficher`

qui agissent sur un fichier de données `telephones.dat`

Format : un numéro et un nom par ligne

exemple

01234578 PUF

98765444 Charlie

application (suite)

```
#!/bin/bash
# tel-afficher
#
nomFichier="telephones.dat"
cat $nomFichier
```

```
#!/bin/bash
# tel-ajouter numero nom
#
nomFichier="telephone.dat"
echo $@ >> $nomFichier
```

```
#!/bin/bash
# tel-chercher nom
#
nomFichier="telephone.dat"
grep $1 $nomFichier
```

Améliorez la présentation avec la commande `dialog`

- `dialog --infobox message hauteur largeur`
- `dialog --textbox nomfichier hauteur largeur`
- `dialog --inputbox message hauteur largeur`

Attention : Avec une “inputbox”, le résultat va sur la sortie d’erreur.

Affectation du résultat d'une commande

Commande et variable

- Ne pas confondre

<code>v=date</code>	affectation de la chaine "date"
<code>date > f</code>	redirection de la sortie vers un fichier
<code>v=\$(date)</code>	affectation de la sortie dans une variable

- Exercice : que fait ceci

```
$cmd > $f
```

?

Exercice : que fait ceci

```
$cmd > $f
```

?

Exemple

```
format="%Y-%M-%d"
```

```
cmd="date +$format"
```

```
f=/tmp/resultat
```

```
$cmd > $f
```

Chaînes et expansion

L'expansion

- se fait dans les chaînes délimitées par `"..."`
- pas dans les chaînes délimitées par `'...'`

Exemple

```
echo 'la variable $USER ' "contient $USER"
```


Cinquième partie V

Les fonctions

Un script peut comporter des **fonctions**, avec des **paramètres positionnels**

Syntaxe

```
function nom-de-fonction  
{  
    commande  
    commande  
    ...  
}
```

Fonctions : exemple

Exemple

```
#!/bin/bash
```

```
function archiver
```

```
{
```

```
    tar -czf "/var/svgd/$1.tgz" "$2"
```

```
}
```

```
archiver photos /home/billaud/photos
```

```
archiver musique /home/billaud/musique
```

Fonctions : avantages

Avantages :

- découpage logique,
- code plus facile à lire
- fonctions réutilisables

Fonctions : exemple

Par défaut, les variables sont communes (globales)

Exemple

```
#!/bin/bash
```

```
destination=/var/svgd
```

```
function archiver
```

```
{
```

```
    tar -czf "${destination}/${1}.tgz" "$2"
```

```
}
```

```
archiver photos /home/billaud/photos
```

```
archiver musique /home/billaud/musique
```

Variables locales

On peut déclarer des **variables locales** dans une fonction

Exemple

```
#!/bin/bash
```

```
destination=/var/svgd
```

```
function archiver
```

```
{
```

```
    local nom="$(basename $1)"
```

```
    tar -czf "${destination}/${nom}.tgz" "$1"
```

```
}
```

```
archiver /home/billaud/photos
```

```
archiver /home/billaud/musique
```

Sixième partie VI

Arithmétique

Let : affectation arithmétique

Exercices

Expansion arithmétique

Let : affectation arithmétique

Let : affectation arithmétique

Syntaxe

```
let VARIABLE=EXPRESSION
```

Exemple à essayer

```
#!/bin/bash
```

```
let somme=$1+$2
```

```
echo $somme
```

Comparer

- `let somme=$1+$2`
- `somme=$1+$2`

Dans une affectation arithmétique, l'expansion des variables est automatique

```
let c=a+b
```

```
let c=$a+$b
```

Exercices

Exercice 1

Ecrire un script qui

- demande l'année de naissance
- affiche l'age

scenario

```
$ exercice1.sh
```

```
Vous êtes né en quelle année ?
```

```
1990
```

```
Vous avez donc 20 ans
```

```
$
```

Exercice 2

Convertir une heure (donnée sous la forme HHMM ou HMM) en nombre de minutes. Assurez-vous que 0900 fonctionne.

scenario

```
$ minutes.sh 1015
```

```
615
```

```
$
```

Écrire un script qui calcule la durée d'un trajet, à partir des heures de départ et d'arrivée sous la forme HHMM

Scénario

```
$ ./duree.sh 630 2215  
1545
```


Expansion arithmétique

Expansion arithmétique

A la place de

```
let surface=hauteur*largeur  
echo la surface du rectangle est $surface m2
```

On peut écrire

```
let surface=hauteur*largeur  
echo la surface du rectangle est $((largeur*hauteur)) m2
```

Septième partie VII

Processus

Définitions

Table des processus

kill

Pilotage des processus

Définitions

Un processus = un programme “qui tourne”

Un programme lancé depuis le shell peut

- tourner en “avant plan” (foreground) : il faut attendre sa fin pour lancer une autre commande
- tourner en “arrière-plan” (background)
- être stoppé

Pour

lancer une commande en avant-plan	xclock
stopper la commande en avant-plan	CTRL-Z
relancer la commande stoppée en avant-plan	fg
relancer la commande stoppée en arrière plan	bg
lancer une commande en arrière-plan	xclock &

Note : Si il y a plusieurs commandes en arrière-plan, commandes

- jobs,
- fg %n,
- etc.

Table des processus

La table des processus

On peut la voir par la commande `ps`, ou `top`, ...

Exemple

```
$ ps
  PID TTY          TIME CMD
 4056 pts/1    00:00:00 bash
 4236 pts/1    00:00:07 xpdf.bin
 4243 pts/1    00:00:10 emacs
 4471 pts/1    00:00:00 xterm
 4613 pts/1    00:00:00 ps
```

- `ps` sans option montre les processus issus du shell
- options intéressantes : `axule...`
- voir aussi `pstree`

kill



La commande “kill”

- Syntaxe : `kill [-signal] num-processus ...`
- Rôle : envoie un **signal** à des processus

Exemple

```
$ xclock -digital -update 1 &  
[6] 4734
```

```
$ ps  
  PID TTY          TIME CMD  
 4056 pts/1    00:00:00 bash  
 4236 pts/1    00:00:07 xpdf.bin  
 4734 pts/1    00:00:00 xclock  
 4739 pts/1    00:00:00 ps
```

```
$ kill -KILL 4734
```

La commande “kill” (suite)

- Par défaut, utilise le signal KILL (9) qui termine le programme.
- le signal STOP arrête un processus
- le signal CONT le relance
- `kill -l` affiche la liste des signaux

Pilotage des processus

Pilotage des processus

- commande `&` lance une commande en arrière-plan
- la variable `$!` contient son numéro de processus
- la variable `$$` = numéro du shell courant

Exemple

```
#!/bin/bash
```

```
mplayer funny-music.mp3 >/dev/null &  
music=$!
```

```
# sauvegardes
```

```
tar czf .....
```

```
# arrêter la musique à la fin
```

```
kill -9 $music
```

Wait

wait nnn attend un processus

Exemple

```
#!/bin/bash
```

```
mplayer funny-music.mp3 >/dev/null &  
music=$!
```

```
# sauvegardes en parallèle
```

```
tar czf archive1.tar ..... &  
svgd1=$!
```

```
tar czf archive2.tar ..... &  
svgd2=$!
```

```
wait $svgd1
```

```
wait $svgd2
```

```
kill -9 $music # arrête la musique
```

Huitième partie VIII

Structure de contrôle : case

Présentation

Exemple

Motifs d'un case

Une application avec des processus

Un exemple de service

Le code principal

Les fonctions

Présentation

Structure de contrôle “case”

Choisit les commandes à exécuter en fonction d'un sélecteur

- semblable au “switch” de C++
- le sélecteur est une chaîne de caractères

Exemple sérieux

```
#!/bin/bash
```

```
# Usage : archiver nom-de-répertoire
```

```
echo "Format = normal gz ?"
```

```
read format
```

```
case "$format" in
```

```
gz)
```

```
    option=z ;   suffixe=tgz   ;;
```

```
normal | "" )
```

```
    option= ;    suffixe=tar   ;;
```

```
*)
```

```
    echo "format '$format' non reconnu" >&2
```

```
    exit 1
```

```
;;
```

```
esac
```

```
prefixe="$(basename "$1")"
```

```
tar -c${option}f "$prefixe.$suffixe" "$1"
```

Exemple

Exemple

```

#                               ;;
# usage :                       chercher)
# tel ajouter num nom          grep "$2" "${nomFichier}"
# tel chercher nom             ;;
# tel voir                     voir)
#                               cat "${nomFichier}"
nomFichier="telephone.dat"      ;;
                                *)
case "$1" in                   echo "Erreur" >&2
ajouter)                       exit 1
    shift                      ;;
    echo $* >> "${nomFichier}" esac

```

Motifs d'un case

Plusieurs motifs pour un même cas

```
case $reponse in
oui | o )
    echo "d'accord"
    ;;
non | n )
    echo "tant pis"
    ;;
esac
```


Motifs d'un case hackers

Utilisation des “jokers” de bash

```
case $reponse in
[oO] [Uu] [iI] | [oO] )
    echo "d'accord"
    ;;
[nN] [oO] )
    echo "tant pis"
    ;;
*)
    echo "quoi ?"
    ;;
esac
```

Une application avec des processus

Un exemple de service

Une commande pour faire apparaître / disparaître une pendule sur le bureau

Usage

- `./pendule.sh start`
- `./pendule.sh stop`
- `./pendule.sh usage`
- `./pendule.sh restart`

Le code principal

Code 2/2

```
case "$1" in
start)
    do_start ;;
stop)
    do_stop ;;
restart)
    do_stop
    do_start ;;
usage)
    print_usage ;;
*)
    print_usage
    exit 1
esac
```

Code 1/2

```
prog=/usr/bin/xclock  
pid_file=/tmp/$USER.pid
```

```
function do_start {  
    $prog &  
    echo $! > $pid_file  
}
```

```
function do_stop {  
    kill -9 $(cat $pid_file)  
}
```

```
function usage {  
    echo "usage: pendule {start|stop|restart|usage}"  
}
```

Neuvième partie IX

Boucle for

Boucle for

Exercice

Exercice (seq)

Exercice (find)

Boucle for

Forme générale

```
for VAR in LISTE
do
    COMMANDE
    COMMANDE
    . . . .
done
```

Boucle avec une variable qui parcourt une liste de mots.

Boucle for, exemples simples

```
for f in *.cc  
do  
    astyle --style=gnu "$f"  
done
```

```
for f in *.cc  
do  
    echo "le fichier $f contient $(wc -l $f) lignes"  
done
```

Exercise

Ecrire une commande qui calcule la somme de ses paramètres (en nombre illimité)

```
$ somme 100 3 20
```

```
123
```

Exercise (seq)

Exercice (seq)

Analyser le script

```
r=1
for i in $(seq 1 $1)
do
    let r*=i
done
```

Exercise (find)

Exercice (find)

Analyser le script

```
for f in $(find ~ -name '*.cc' -ctime -7)
do
    ls -l "$f"
done
```

- rôle de `find ~`
- rôle de `find ~ -name '*.cc'`
- rôle de `find ~ -name '*.cc' -ctime -7r`
- pourquoi les guillemets dans `ls` ?

Dixième partie X

Décisions

Code de retour (exit status)

if-then-else-fi

La commande test

Enchaînement conditionnel

if-then-elif-else, forme générale

Code de retour (exit status)

Code de retour (exit status)

- Le **code de retour** d'un programme indique si il s'est bien terminé.
- Le code de retour de la dernière commande est dans la variable \$?
- Par convention : 0 = OK.

Exemple

```
$ ls -ld /tmp
drwxrwxrwt 10 root root 12288 déc 22 17:32 /tmp
$ echo code de retour = $?
code de retour = 0
```

```
$ ls -l qdsdqd
ls: ne peut accéder qdsdqd: Aucun fichier ou
répertoire de ce type
$ echo code de retour = $?
code de retour = 2
```

Les codes de retour des commandes sont décrits dans le manuel

Exemple : man ls

...

Exit status is 0 if OK, 1 if minor problems,
2 if serious trouble.

- Par défaut, un script retourne le code de sa dernière commande
- Il peut retourner un code spécifique par
`exit [code]`

Exemple

```
#!/bin/bash  
echo "Something strange happened"  
exit 42
```

if-then-else-fi

La structure de contrôle `if then else fi` utilise le code de retour d'une commande

Exemple

```
#!/bin/bash
# usage:
#   compiler.sh prefixe
#
if g++ -o "$1" "$1.cc"
then
    echo "la compilation s'est bien passée"
else
    echo "il y a eu un problème" >&2
fi
```

La commande test

Le code de retour du programme test dépend d'une **condition**.

Exemple :

```
test -f nomFichier
```

indique si le fichier existe.

Application

fichier "compiler.sh"

```
#!/bin/bash
if test -f "$1.cc"
then
    if g++ -Wall -o "$1" "$1.cc"
    then
        echo "Compilation terminée"
    else
        echo "Erreur de compilation">&2
        exit 1
    fi
else
    echo "Erreur: pas de fichier $1.cc" >&2
    exit 2
fi
```

Autre notation : [condition]

Autres tests :

[-d nom]	# nom désigne un répertoire
[chaine1 = chaine2]	# comparaison de chaines
[chaine1 != chaine2]	
[chaine1 \< chaine2]	
[chaine1 \> chaine2]	
[nombre1 -eq nombre2]	# comparaison de nombres
[nombre1 -ne nombre2]	
[nombre1 -le nombre2]	# less or equal
[nombre2 -ge nombre2]	# greater or equal
	# voir aussi -lt et -gt

Ecrire une commande qui affiche le maximum de deux paramètres.

Scénario

```
$ max.sh 37 421
```

```
421
```

Enchaînement conditionnel

Syntaxe

COMMANDE1 && COMMANDE2

COMMANDE1 || COMMANDE2

Exécute la seconde commande seulement si la première a réussi (&&) ou échoué (||)

Exemple

```
g++ prog.cc && echo OK
```


Application

```
#!/bin/bash
```

```
max=$1
```

```
test $2 -ge $max && max=$2
```

```
echo $max
```

if-then-elif-else, forme générale

if-then-elif-else, forme général

Syntaxe

```
if COMMANDES
then
    COMMANDES
[ elif COMMANDES
  then
    COMMANDES
]...
[ else
    COMMANDES
]
fi
```

Onzième partie XI

Boucle while

Boucle while

Boucle while-read

break

Qualité du shell

Boucle while

- Syntaxe

```
while COMMANDE  
do  
    COMMANDES  
done
```

Exemple

Que fait ce script ?

```
#!/bin/bash
```

```
i=1
```

```
f=1
```

```
while test $i -le $1
```

```
do
```

```
    let f=f*i
```

```
    let i++
```

```
done
```

```
echo $f
```


Boucle while-read

Pour traiter le contenu d'un fichier, ligne par ligne.

```
while read numero nom
do
    printf "| %12s | %-30s |\n" $numero $nom
done < annuaire.txt
```

break

L'instruction break

permet de sortir d'une boucle

```
while true
do
    . . . .

    echo "voulez-vous arrêter ?"
    read reponse
    test "${reponse}" = oui  && break

    . . .
done
```

Qualité du shell

Template de base

```
#!/bin/bash
```

```
set -o errexit # Exit if command failed.
```

```
set -o pipefail # Exit if pipe failed.
```

```
set -o nounset # Exit if variable not set.
```

```
# Remove the initial space and instead use '\n'.
```

```
IFS=$'\n\t'
```

<https://google.github.io/styleguide/shell.xml>

`https://www.shellcheck.net/`

```
$ shellcheck myscript
```

```
In myscript line 5:
```

```
eof
```

```
^-- SC1118: Delete whitespace after the here-doc end token
```